Tu código CSS puede ser más limpio, flexible y reutilizable.

sergio-lopes
30/04/2021

Los buenos programadores aprenden buenas prácticas de código desde una edad temprana. Encapsulación, buenos nombres de variables y métodos, orientación a objetos y otros conceptos clásicos están en la punta de la lengua. A pesar de esto, veo a muchos buenos programadores que hacen un mal código de front-end. ¿Mi opinión? **Hacer un buen código de front-end es más difícil de lo que parece**.

Escribir un *buen JavaScript* suele ser más fácil. Como es un lenguaje de programación similar a los demás, conseguimos mapear bien los conceptos de encapsulación y orientación a objetos (a pesar de la dificultad de los prototypes).

Pero con HTML y CSS, comienza a complicarse. ¿Cómo tener un CSS más reutilizable? Algunas buenas prácticas para desarrolladores front-end:

No sobreescribe tus propias reglas CSS.

Has estado ahí: pon un `float:left` en un selector y luego necesitas limpiarlo con `float:none` en un caso particular. O incluso reescribir un `width:auto` y cosas asi. **Mala práctica**.

Además de hacer que escribas la misma propiedad dos veces, no haces uso de los estándares del navegador (en este caso, que todos `float` ya es none). Evita esto invirtiendo sus selectores: deja solo los elementos con los que realmente necesita flotar con `float:left` y el resto hereda el valor predeterminado none y listo.

Un escenario en el que veo que esto sucede mucho es cuando se adaptan sitios para pantallas más pequeñas con consultas de medios. En la pantalla grande, los elementos flotan en todas partes, pero en un teléfono celular pequeño decide apilar los elementos y eliminar el `float`:

```
/* desktop */
.botao-magico { float: right; }

/* mobile */
@media (max-width: 600px) {
    .botao-magico { float: none; }
}*
```

En este caso de consultas de medios, por cierto, esto sólo sucedió porque usamos la estrategia de escribir los estilos del escritorio primero y luego sobreescribirlos con los

estilos para dispositivos más pequeños. Llaman a esta estrategia **desktop-first** y este ejemplo es un gran argumento para que no hagas eso y prefieras **mobile-first** en tu CSS:

```
/* mobile */
/* ¡nada aquí! el estándar es float:none */

/* desktop */
@media (min-width: 600px) {
    .boton-magico { float: right; }
}
```

Abusa de clases para reutilizar código CSS. Evita ID.

ID son muy inflexibles: sólo pueden aparecer una vez en la página y un elemento no puede tener más de un ID (el elemento es un *singleton*). Esto hace que ID sea el villano a la hora de reutilizar el código.

Tú deberías **siempre usar clases en tu CSS**. Esto te permite aplicar el mismo estilo a más de un elemento diferente sin replicar reglas. Tu código se vuelve más corto y más limpio.

Las clases CSS nos permiten el tipo de reutilización que nos brinda la herencia en los lenguajes orientados a objetos. Cree una jerarquía de clases para reutilizar estilos comunes incluso en diferentes elementos.

Por ejemplo: tienes dos elementos en la página (`.menu` y `.noticias`) que tienen diferentes funciones y efectos visuales. Pero los dos siguen la base visual del sitio, que es tener un fondo gris, con un borde redondo y una sombra. En lugar de copiar propiedades, crea una nueva clase base para estos elementos:

```
<div class="box menu">...</div>
<div class="box noticias"></div>
    .box {   background: #ccc;
    border-radius: 5px;
    box-shadow: 2px 2px 2px black;
}
```

Es preferible que crees esta clase nueva `.box` que escribir CSS usando las clases de los otros dos elementos `.menu`, `.noticias` { ... }. El día que aparezca un tercer box,

simplemente aplica la clase, sin interferir en el CSS.

No luches con la especificidad de CSS.

Hablando de clases y reglas primordiales, las cosas empeoran aún más con las reglas de especificidad de CSS. El selector de ID tiene prioridad sobre la clase, que tiene prioridad sobre el tagname.

Otra razón para que evites los ID en CSS es que no puedes sobrescribir sus propiedades con sus clases genéricas reutilizables. Necesitarías otra ID para sobrescribir un selector de ID.

No luches con especificidad. Mantén la especificidad de tus selectores al mismo nivel usando clases siempre para estilizar. Son más fáciles de componer. Y nunca uses `!important` para forzar un estilo.

Los números mágicos son una mala señal en CSS.

Cada vez que escribes un `margin-top:37px` o un `width:381px` se muere una cría de foca. Evita los números mágicos en tu CSS que son **calculados arbitrariamente** y **seguramente se romperán** si el texto es un poco más largo o si tu menú tiene una entrada adicional.

Peor aún, ese número mágico para alinear algo (como uno `top:-1px`) que sin duda tiene en cuenta la representación en un determinado navegador y se interrumpirá en otro navegador o si el icono cambia. Esto hace que tu CSS sea inutilizable y requiere un mantenimiento constante. ¿Quieres alinear un icono con el texto? Aprende cómo usar el vertical-align:middle (a pesar de que no funciona de la forma que crees que funciona).

Utiliza unidades flexibles.

Los números mágicos se pueden evitar en muchos casos usando unidades relativas como porcentajes. Si tienes una página de `940px` de ancho y necesitas dividirla en 5 columnas, no escribas `width:188px` porque nunca recordarás de dónde vino ese valor. Es preferible `width:20%` que muestra de una manera más obvia que es 1/5 de la página.

Y, por supuesto, si puedes, haz todas tus unidades de layout con porcentajes. Esto hará que tu diseño sea flexible y no dependerá del tamaño del navegador. **La Web es una media**

elástica y limitar tu página a píxeles estáticos es transformar la Web en una media más limitada como la impresa.

Para elementos tipográficos o afectados por la tipografía, utiliza em como medida flexible.

Evita acoplar el estilo CSS a tags específicas de HTML.

Escribir el nombre tag en el medio del CSS hace que tu estilo se combine con la estructura de HTML. Si el HTML cambia, tu estilo se rompe. Podemos evitar esto creando más clases y disminuyendo el acoplamiento entre HTML y CSS.

Si tienes un menú como una lista llena de elementos, no uses `.menu li` en CSS. Es preferible crear una clase específica `.menu-item` y aplicarla en HTML. Podrás cambiar tu HTML más tarde sin ningún problema.

Incluso cuando usas elementos más estandarizados, como `<header>` o `<h1>`, una buena práctica es usar clases para estilizar:

```
<header class="tope">
    <h1 class="llamada-principal">¡Compre ahora mismo!</h1>
</header>
```

Todo esto te deja mucha más libertad para cambiar el HTML por razones de semántica y contenido, sin afectar el estilo CSS.

Dale buenos nombres a tus clases

Y dado que buena parte de las prácticas pasa por crear nuevas clases, es bueno saber nombrarlas correctamente. La regla obvia, ya sea que ya la esté siguiendo, es crear **nombres legibles y fáciles de entender**. Usar `.panel-principal` al revés de `.pnlPri`.

Pero eso no es todo. Los nombres de las clases son el puente entre tu HTML y tu CSS. Se escribirán en el medio del contenido HTML de la página y, por lo tanto, **deben tener semántica de contenido y no de visual**.

Eso significa que no deberías crear clases llamadas `.box-lateral` o `.titulo-azul` o `.panel-derecha`. Si tu box no es lateral o el título cambia de color o el panel cambia de lado, tu clase pierde el significado.

Evita esto creando clases con nombres de contenido, como `.panel` o `.llamada-principal`. Y, por supuesto, si, eventualmente, tienes un `.titulo-azul` en tu código, no dejes la clase con este nombre el día que el color cambia a rojo; ¡cambia el nombre de la clase!

Documenta bien las excepciones.

Cada regla tiene excepciones. A veces, pondrás una ID o tagname en tu CSS. O necesitarás un `!important` en un lugar raro. O aquel número mágico que no recordarás después como calculas.

Esto debe ser una excepción, pero no abuses. Y, siempre que hagas esto, documenta el motivo de esa elección directamente en el CSS.

```
/* estructura esperada para el menú:
    <ul class="menu">
        <li><a href=""></a>
        <li><a href=""></a>
    </ul>
*/
.menu { ... }
.menu li { ... }
.menu a { ... }

/* usando ID porque el widget de facebook obliga */
#facebook-like iframe {
    width: 100% !important; /* important para sobreescribir css inline de widg
}
```

Más

Por supuesto que hay muchas otras buenas prácticas. Debes identificar bien tu código, ordenar las propiedades de manera lógica y preocuparte con performance. Puedes utilizar preprocesadores, frameworks y técnicas avanzadas (como OOCSS). En los cursos de front-end de Alura Latam, hablamos de varias de estas técnicas y con muchos ejemplos prácticos.

Lo importante es tener cuidado especial para que tu CSS sea siempre lo más fácil, reutilizable y flexible posible. Esto ayudará a tu vida diaria y la de los demás desarrolladores de tu equipo.

Puedes leer también:

- Creación de componentes CSS con el estándar BEM
- Empezando a organizar tu CSS
- Centrar un elemento con CSS

ARTÍCULOS DE TECNOLOGÍA > FRONT END

- Acceso a todo el contenido de la plataforma por 6 meses

¡QUIERO EMPEZAR A ESTUDIAR!

Paga en moneda local en los siguientes países

ANUAL

US$79,90

un solo pago de US$79,90

- 218 cursos
- Videos y actividades 100% en Español
- Certificado de participación
- Estudia las 24 horas, los 7 días de la semana
- Foro y comunidad exclusiva para resolver tus dudas
- Acceso a todo el contenido de la plataforma por 12 meses